Precise Requirements Engineering for Model Transformations

S. Yassipour Tehrani, K. Lano

Dept. of Informatics, King's College London

Abstract. Requirements engineering is an essential process in the development of effective software systems, and it is the basis for subsequent development processes. In my PhD research I intend to identify techniques for the systematic requirements engineering of model transformations, taking account of the specific characteristics of different categories of model transformations.

1 Introduction

Requirements engineering has been a relatively neglected aspect of model transformation development: the emphasis in transformation development has been upon specification and implementation. The failure to explicitly identify requirements may result in developed transformations which do not satisfy the needs of the transformation users. Problems may arise because implicitly-assumed requirements have not been explicitly stated, for example, that a migration or refactoring transformation should preserve the semantics of its source model in the target model, or that a transformation is only required to operate on a restricted range of input models. Without thorough requirements elicitation, important requirements may be omitted from consideration, resulting in a developed transformation which fails to achieve its intended purpose.

As [14] argues, "we are far from making the writing of model transformations an established and repeatable technical task". The software engineering of model transformations has only recently been considered in a systematic way, and most of this work (eg., [2, 7, 9]) has focussed upon design and verification rather than upon requirements engineering.

2 Model Transformation Requirements

Requirements for a system are generally divided into two main categories: functional requirements, which identify what functional capabilities the system should provide, and non-functional requirements, which identify quality characteristics expected from the developed system, and restrictions upon the development process itself.

The functional requirements of a model transformation $\tau : S \rightarrow T$ which maps models of a source language S to a target language T are typically defined by a set of *mapping requirements* considering different cases of structures

and elements within source models $s : S$. For refactoring transformations there may be locally-defined *refactoring requirements* defining how particular cases of structure within a model should be rewritten. In addition, assumptions about the data of the input model should be identified as part of the functional requirements.

It can be observed in many published examples of model transformations that the initial descriptions of their intended functional behaviour is in terms of a *concrete syntax* for the source and target languages which they operate upon. For example, in [5], the three key effects of the transformation are expressed in terms of rewritings of UML class diagrams. In [15], the transformation effects are expressed by parallel rewritings of Petri Nets and statecharts. In general, specification of the intended functionality of the transformation in terms of concrete syntax rules is more natural and comprehensible for the stakeholders than is specification in terms of abstract syntax, however this form of description has the disadvantage that it may be imprecise: there may be significant details of models which have no representation in the concrete syntax, or there may be ambiguities in the concrete syntax representation. Therefore, conversion of the concrete syntax rules into precise abstract syntax rules is a necessary step as part of the formalisation of the requirements.

In addition to local functional requirements, there may be functional requirements which specify that some global property of the model is achieved. In particular, refactoring transformations may be aimed at improving some quality measure of the model, e.g., to remove redundant elements from the model or otherwise to improve its structure, *model quality improvement* requirements.

Figure 1 shows a taxonomy of functional requirements for model transformations, based on our experience of transformation requirements.

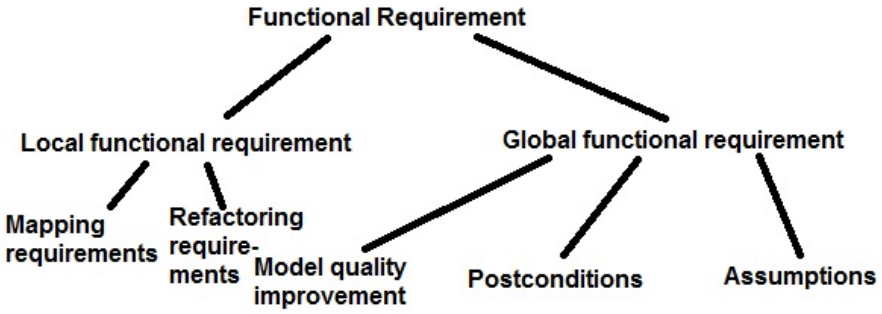

Fig. 1. Taxonomy of functional requirements for model transformations

There may be a wide range of different non-functional requirements for a system [16], in categories such as Quality of service, Compliance, Development

constraint, etc. Figure 2 shows a general decomposition of non-functional requirements for model transformations. The quality of service categories correspond closely to the software quality characteristics identified by the IEC 9126 software quality standard [3, 4].

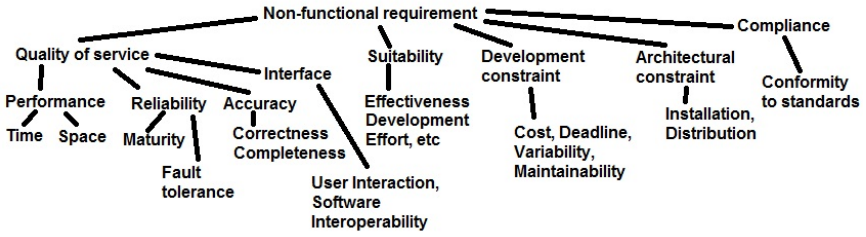

Fig. 2. Taxonomy of non-functional requirements (adapted from [16])

Quality of service characteristics can be further detailed. For Time performance, there may be upper bound requirements on execution time for specific model sizes, maximum capability requirements on sizes of models which should be processed within a reasonable time limit, and requirements on the growth of execution time with input model size. Reliability can be refined into subcharacteristics such as Maturity and Fault Tolerance.

For the *Accuracy* characteristic, we can identify subcharacteristics of *Correctness* and *Completeness*. Correctness requirements can be further subdivided into the following specific forms [8]: *Syntactic correctness*; *Termination*; *Confluence*; *Model-level semantic preservation*; *Invariance* (that certain properties *Inv* should be preserved as true during the execution of the transformation τ).

2.1 Requirements Formalisation

Mapping functional requirements can be formalised in a MT-language-independent manner by using diagrams or predicates at the abstract syntax level. There should be a direct correspondance between the concrete syntax elements in the informal/semi-formal expression of the requirements, and the abstract syntax elements in the formalised versions. Similarly, postconditions and assumptions can be formalised by expressing them as predicates or diagrams at the concrete syntax level. Model quality improvement requirements can be formalised by defining a specific quantitative quality measure which should be improved. For quality of service requirements, specific quantifiable measures for the properties of interest should be identified, and precise bounds on the permitted values of these measures (or ranges of acceptable values) specified.

3 Application of Requirements Engineering Methods

There are a wide range of existing requirements engineering techniques which could be applicable to transformation requirements engineering. For example, i^* [17] can be used to establish connections between organisation goals and the functional requirements of the intended transformation system. *REMAP* [10] uses goals as a basis for an argumentation process which selects a design solution to satisfy the goals. Traces are established from requirements to the design objects which are intended to achieve the requirements. The NFR framework can be used to identify design alternatives to meet the non-functional requirements of a transformation system [1]. An approach which seems particularly well-aligned with requirements engineering of model transformations is KAOS [16], which supports requirements elaboration using temporal logic. The 'Cease' goal pattern of KAOS fits the usual case of refactoring transformations which must remove structures of particular kinds in the model. Each local refactoring requirement can be expressed by such a goal pattern, asserting that each occurrence of a condition φ which should be removed will eventually be removed, and will not be reintroduced:

$$\textit{model elements } x1,\ ...,\ xn \textit{ satisfy property } \varphi \ \Rightarrow\ \diamond\,\Box(x1, ..., xn \textit{ do not satisfy } \varphi)$$

Transformation invariants can be expressed using the 'Maintain' goal pattern:

$$Asm \ \Rightarrow\ \Box(Inv)$$

General postconditions can be expressed using the 'Achieve' pattern:

$$Asm \ \Rightarrow\ \diamond\,(Post1 \wedge ... \wedge Postm)$$

Formalised requirements in temporal logic could then be checked for particular implementations using model-checking techniques, as in [11].

4 Published examples of transformation requirements

We consider two published examples of requirements of transformations: the model migration case study of [12] and the refactoring case study of [5]. Both documents specified the requirements of transformations which were used for comparative analysis of different MT approaches. The results of the analysis were also published.

The model migration example concerns the migration of UML 1.4 activity diagram models to UML 2.2 activity diagrams [12]. The requirements consist of:

Functional requirements: the transformation should successfully migrate one example activity diagram which includes all of the core elements of UML 1.4 activity diagrams.

Non-functional requirements: size and comprehensibility of the specification should be optimised.

It can be seen that many of the categories of requirements in our catalogue of transformation requirements are missing for this case study. In particular: what assumptions can be made upon the input model; model-level semantic preservation; confluence; reliability; time performance. The required functionality is only indicated by one examplar case of a source model and its intended target.

The poor quality of the published solutions [13] may result in part from these incomplete requirements:

- The highest score for correctness for the 9 solutions was 5.5 out of 10.
- No solution provided a proof of model-level semantic preservation. The proposed migration strategy would lose semantic information (the action performed in action states).
- The issue that some valid UML 1.4 activity diagrams are not valid as UML 2.2 activities (when directly translated according to the example given) was not addressed by any solution [6].

All of these aspects would hinder the practical use of transformations for this migration problem.

The refactoring example operates on UML class diagrams to remove cases where all subclasses of a given class have an attribute with a common name and type. The requirements of the case study are described in some detail in [5]:

Functional requirements: the assumptions are precisely defined; three intended transformation steps (refactoring requirements) are described in text and concrete syntax diagrams.

Non-functional requirements: invariance of the assumptions is specified. Effectiveness is specified in terms of minimising the number of new classes created.

Missing are requirements for model-level semantic preservation, for model quality improvement, and for reliability, extensibility, efficiency and comprehensibility.

The outcomes of the case study were that:

- None of the five published solutions in [5] provide a proof of model-level semantic preservation. Some solutions do not achieve such preservation, because they use a different set of rules to those indicated in the functional requirements.
- None of the solutions achieve more than a neutral measure of usability or extensibility.
- Only 2 solutions have a practical efficiency in processing large models.

5 Conclusions and Future Work

My PhD research has as its research objectives: to define a requirements engineering process for model transformations; to define taxonomies of functional and non-functional requirements categories for model transformations; to define

goal-modelling techniques for transformations; to analyse industrial use of requirements engineering by transformation developers, to identify the significant issues and support needed for RE of model transformations. The result of the PhD research will be a systematic RE process for transformations, and possibly tool support for recording and tracing transformation requirements, which will help to ensure that developers systematically consider all necessary requirements and that these are all formalised, validated and verified correctly.

References

1. L. Chung, J. Mylopoulos, E. Yu, *From Object-oriented to Goal-oriented Requirements Analysis*, Communications of the ACM, January 1999.
2. E. Guerra, J. de Lara, D. Kolovos, R. Paige, O. Marchi dos Santos, *transML: A family of languages to model model transformations*, MODELS 2010, LNCS vol. 6394, Springer-Verlag, 2010.
3. ISO/IEC, *ISO/IEC 9126-1, Software engineering – Product quality – Part 1: Quality Model*, 2001.
4. ISO/IEC JTC1/SC7, *ISO/IEC 25010, Software product Quality Requirements and Evaluation (SQuaRE)*, 2007.
5. S. Kolahdouz-Rahimi, K. Lano, S. Pillay, J. Troya, P. Van Gorp, *Evaluation of model transformation approaches for model refactoring*, Science of Computer Programming, 2013, http://dx.doi.org/10.1016/j.scico.2013.07.013.
6. K. Lano, S. Kolahdouz-Rahimi, *Model migration transformation specification in UML-RSDS*, TTC 2010.
7. K. Lano, S. Kolahdouz-Rahimi, *Model-driven development of model transformations*, ICMT 2011, 2011.
   Kolahdouz-Rahimi, transformations,
8. K. Lano, S. Kolahdouz-Rahimi, T. Clark, *Comparing verification techniques for model transformations*, Modevva workshop, MODELS 2012.
9. K. Lano, S. Kolahdouz-Rahimi, *Constraint-based specification of model transformations*, Journal of Systems and Software, February 2013.
10. B. Ramesh, V. Dhar, *Supporting systems development by capturing deliberations during requirements engineering*, IEEE Transactions on Software Engineering, vol. 18, no. 6, 1992, pp. 498–510.
11. A. Rensink, A. Schmidt, D. Varro, *Model checking graph transformations: A comparison of two approaches*, ICGT 2004, LNCS vol. 3256, 2004.
12. L. Rose, D. Kolovos, R. Paige, F. Polack, *Model migration case for TTC 2010*, Transformation Tool Contest, 2010.
13. L. Rose, M. Herrmannsdoerfer, S. Mazanek, P. Van Gorp et al., *Graph and Model Transformation Tools for Model Migration*, SoSym, 2012.
14. B. Selic, *What will it take? A view on adoption of model-based methods in practice*, Software Systems Modeling, 11:513–526, 2012.
15. P. Van Gorp, L. Rose, *The Petri-Nets to Statecharts Transformation Case*, TTC 2013, EPTCS, 2013, pp. 16–31.
16. A. Van Lamsweerde, *Requirements Engineering: from system goals to UML models to software specifications*, Wiley, 2009.
17. E. Yu, *Towards modelling and reasoning support for early-phase requirements engineering*, proceedings 3rd IEEE Symposium on Requirements Engineering, 1997, pp. 226–235.